John von Neumann Institute for Computing

Efficient Parallel Simulations in Support of Medical Device Design

Marek Behr, Mike Nicolai, Markus Probst

published in

Parallel Computing: Architectures, Algorithms and Applications ,
C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr, F. Peters (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **38**, ISBN 978-3-9810843-4-4, pp. 19-26, 2007.
Reprinted in: Advances in Parallel Computing, Volume **15**,
ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

© 2007 by John von Neumann Institute for Computing
Permission to make digital or hard copies of portions of this work for personal or classroom use is granted provided that the copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise requires prior specific permission by the publisher mentioned above.

http://www.fz-juelich.de/nic-series/volume38

Efficient Parallel Simulations in Support of Medical Device Design

Marek Behr, Mike Nicolai, and Markus Probst

Chair for Computational Analysis of Technical Systems (CATS)
Center for Computational Engineering Science (CCES)
RWTH Aachen University
52056 Aachen, Germany
E-mail: {behr, nicolai, probst}@cats.rwth-aachen.de

A parallel solver for incompressible fluid flow simulation, used in biomedical device design among other applications, is discussed. The major compute- and communication-intensive portions of the code are described. Using unsteady flow in a complex implantable axial blood pump as a model problem, scalability characteristics of the solver are briefly examined. The code that exhibited so far good scalability on typical PC clusters is shown to suffer from a specific bottleneck when thousands of processors of a Blue Gene are employed. After resolution of the problem, satisfactory scalability on up to four thousand processors is attained.

1 Introduction

Parallel computing is enabling computational engineering analyses of unprecedented complexity to be performed. In this article, experiences with parallel finite element flow simulations supporting the development of implantable ventricular assist devices in the form of continuous-flow axial pumps are reported. As an alternative to pulsatile-type devices, these pumps offer simplicity and reliability needed in long-term clinical applications. Their design however poses continuing challenges[1].

The challenges can be itemized as high shear stress levels, flow stagnation and onset of clotting, and loss of pump efficiency. Elevated shear stress levels are particularly evident in mechanical biomedical devices. Biorheological studies[2] point out a number of platelet responses to shear: adhesion, secretion, aggregation (i.e. activation), and finally, hemolysis (i.e. damage). These responses are dependent on both the level and duration of elevated shear at the platelet surface, not unlike the response to chemical factors. The primary response of the red blood cells is hemolysis[3], again dependent on dose and time.

The distribution of the shear stress levels in a complex flow field in a rotary blood pump chamber as well as the duration of the blood cells' exposure to these pathological conditions are largely unknown. Device designers are often compelled to make decisions about the details of pump configuration guided only by the global, time- and space-averaged, indicators of the shear stress inside the pump, such as the hemolysis observations made on the exiting blood stream. This challenge of detailed analysis and reduction of shear stress levels while maintaining pump efficiency as well as the need to pinpoint any persistent stagnation areas in the flow field motivates the current computational work.

In the following sections, the governing equations and their discretization are briefly discussed in Section 2, the parallel implementation described in some detail in Section 3, and parallel simulations of a complex blood pump configuration, with emphasis on scalable performance on the Blue Gene platform, are reported in Section 4.

2 Model and Discretization

2.1 Governing Equations

The blood is treated here as a viscous incompressible fluid, governed by the 3D unsteady Navier-Stokes equations. The primary variables are the velocity and pressure. No-slip Dirichlet boundary conditions are applied on most device surfaces, and stress-free parallel flow boundary conditions are imposed on the inflow and outflow surfaces.

2.2 Finite Element Formulation

The governing equations are discretized using a stabilized space-time Galerkin/Least-Squares (GLS) formulation[4]. This GLS stabilization counteracts the natural instabilities of the Galerkin method in the case of advection-dominated flow fields, and it bypasses the Babuska-Brezzi compatibility condition on the function spaces for the velocity and pressure fields, allowing both fields to use e.g. equal order basis functions. The Deforming Spatial Domain/Stabilized Space-Time (DSD/SST)[5] formulation is utilized, with equal-order linear interpolation for both fields. The motion of the computational domain boundaries is accommodated by a deforming boundary-fitted mesh. The space-time formulation naturally accounts for the deformation of the mesh, giving rise to a scheme comparable to the Arbitrary Lagrangian-Eulerian (ALE)[6] methods.

2.3 Mesh Update

The simulation of fluid flows in the presence of large but regular deformations, such as straight-line translations or rotations can be accomplished in the context of space-time finite element techniques used here by using the Shear-Slip Mesh Update Method (SSMUM) proposed earlier[7]. This method allows the elements in a thin zone of the mesh to undergo 'shear' deformation that accommodates the desired displacement of one part of the domain with respect to another. This zone is re-meshed frequently via regeneration of element connectivity. The idea behind the special-purpose mesh design is the matching of the node positions for the old (deformed) and new (good-quality) meshes, such that projection is never necessary. Since only a small part of the overall connectivity is being regenerated, the computational cost is greatly reduced.

3 Parallel Implementation

The structure of the XNS code—eXperimental Navier-Stokes finite element solver used in authors' group—resembles the one previously described in the context of the Connection Machine[8] and PC clusters[9]; the most significant difference has been replacement of element-by-element matrix storage by block-sparse-row matrix storage. The major computation stages are recalled in the Box 1.

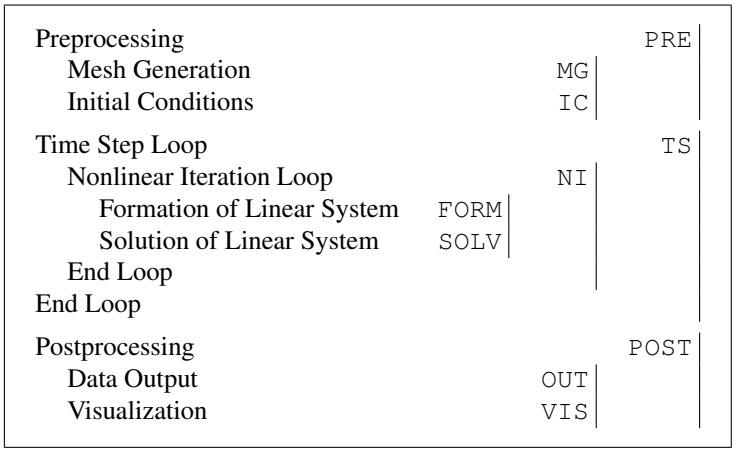


Figure 1. Computational stages of the simulation code.

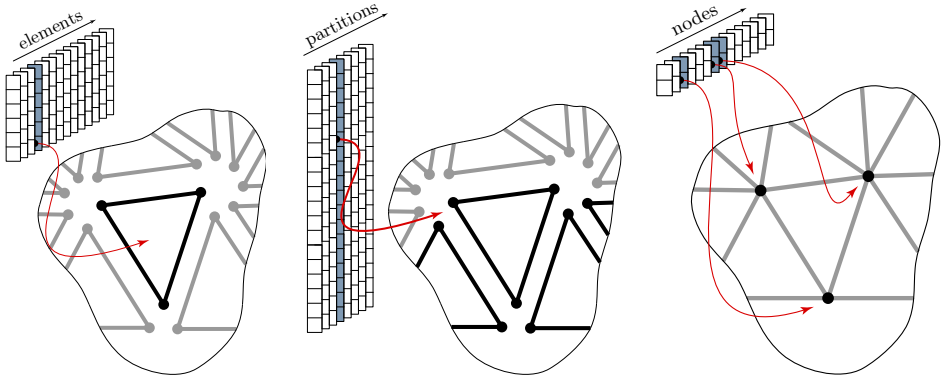
For large-scale unsteady simulations that are the focus here, the operations confined to the nonlinear iteration loop and the time-step loop are contributing most to the overall computational cost. Therefore, only the FORM and SOLV stages will be discussed.

3.1 Data Distribution

Most XNS data arrays fall into two simple categories: partition-based and node-based. In the partition-based arrays, also referred to as PART-type arrays, the last[a] dimension spans the number of nodes in a partition, which corresponds to a contiguous physical subdomain. In the node-based arrays, also referred to as NODE-type arrays, the last dimension spans the number of mesh nodes. Since some nodes are shared between partitions, the overall number of nodes in PART arrays is greater than the overall number of nodes in NODE arrays. A third storage model is useful conceptually, although not necessarily used in its entirety: an ELEM-type array is assumed to store data for all the elements belonging to a single partition. The remaining array dimensions are typically small, and correspond to number of nodes per element, number of degrees of freedom per node, etc. These categories are illustrated in Figs. 2(a), 2(b) and 2(c).

In the distributed-memory implementation, the mesh is partitioned into contiguous subdomains with the aid of the METIS graph partitioning package. Each subdomain and its set of mesh elements is then assigned to a single processing element (PE). The mesh nodes are then distributed in such a way that most nodes which are interior to a subdomain, are assigned to the PE which holds elements of the same subdomain. Nodes at a subdomain boundary are randomly assigned to PEs that store data for elements sharing that boundary. Consequently, contiguous subsets of either the ELEM-type or NODE-type arrays correspond to contiguous sets of elements and nodes, and are stored in local PE memory. The inter-PE communication is thus limited to subdomain boundaries.

[a] Fortran storage sequence is assumed here; a C implementation would use first dimension instead.



(a) element-based (b) partition-based (c) node-based

Figure 2. Storage modes for finite element data.

3.2 Linear System Formation

In the FORM stage, the element-level stiffness matrices and residual vectors are computed through numerical integration element-by-element, with input from various PART-type arrays, such as nodal coordinates, previous flow-field values, and fluid properties which are localized as needed into ELEM-type structure, and the result is assembled immediately into a PART-type matrix and residual at the partition level. This computation does not involve *any* data dependences between different partitions and is in general quite simple to parallelize or vectorize. For the partition-level matrices, Block Sparse Row (BSR) storage[10] is used, with square blocks corresponding to multiple degrees of freedom associated with each node. Such simple storage structure is made possible by equal-order interpolation for the velocity degrees of freedom *and* the pressure.

3.3 Linear System Solution

It is the SOLV phase that presents most complex data-dependence issues and obstacles to parallelization. For most time-dependent problems, the restarted GMRES iterative solver with diagonal or block-diagonal preconditioning is employed. The GMRES stages can be classified into three categories:

- Operations involving complex data dependencies, but carrying insignificant computational cost. The solution of the reduced system falls into that category; such operations can be performed serially on a parallel machine, or even redundantly repeated on all available PEs.
- Operations involving only NODE-type data. Most stages of GMRES, including the Gramm-Schmidt orthogonalization and the preconditioning step fall into this category. These portions are again trivial to parallelize or vectorize, with the only data dependence arising from the need to compute dot products or norms of the NODE-type arrays. These reduction operations are typically provided by the run-time system, e.g., by MPI's MPI_ALLREDUCE call.

- Operations involving both PART and NODE-type data. Barring the use of more complex preconditioners, the only example of such an operation is the matrix-vector product, described below.

In the matrix-vector product, NODE-type arrays represent vectors that must be multiplied by a matrix stored in a PART-type array. In order to avoid the assembly of the global stiffness matrix, the matrix-vector product is performed in the following steps:

1. GMRES:GATHER: localization, or *gather*, of the NODE-type vector into PART-type vector,

2. GMRES:MATVEC: matrix-vector product of that vector and the partition-level contributions to the global stiffness matrix, producing another PART-type vector,

3. GMRES:SCATTER: assembly, or *scatter*, of that vector into NODE-type result.

This is illustrated in Fig. 3.

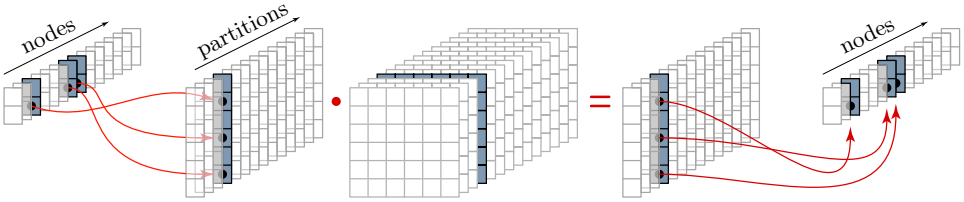


Figure 3. Outline of the matrix-vector product.

The matrix-vector product involving the partition-level stiffness matrix clearly does not involve any data dependencies. In addition, this task is trivially split between multiple PEs in a distributed-memory implementation. On the other hand, the gather and scatter operations involve many-to-one and one-to-many data transfers, and involve communication at the subdomain boundaries in a distributed-memory implementation.

4 Results

The methodology described above is applied to the problem of analysis of blood flow in an axial ventricular assist device—the MicroMed DeBakey LVAD. This pump consists of the flow straightener, a six-bladed impeller, and a six-bladed diffuser inside a cylindrical housing (2.4 cm diameter). The simulations must explore a range of impeller speed, from 7500 to 12000 rpm, and various pressure conditions. The blood viscosity is taken as 0.04 Poise, currently neglecting viscoelastic effects.

An example finite element mesh for this pump configuration has 5,098,349 tetrahedral elements and 1,726,416 space-time nodes, shown in Fig. 4. Two disc-like SSMUM layers, with a total of 30,600 elements, separate the rotating impeller mesh from the stationary straightener and diffuser meshes. The simulations must typically cover 1000 time steps

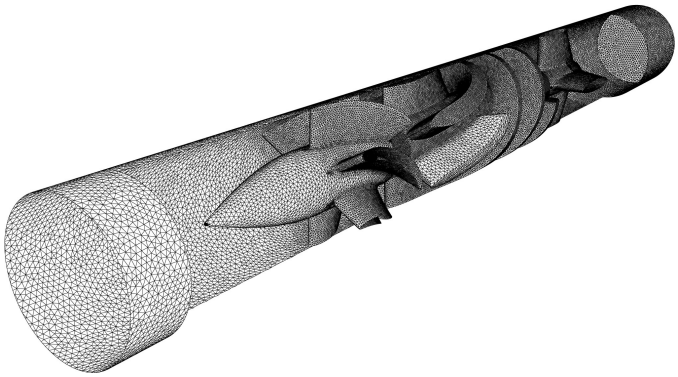


Figure 4. Axial blood pump simulation: computational mesh

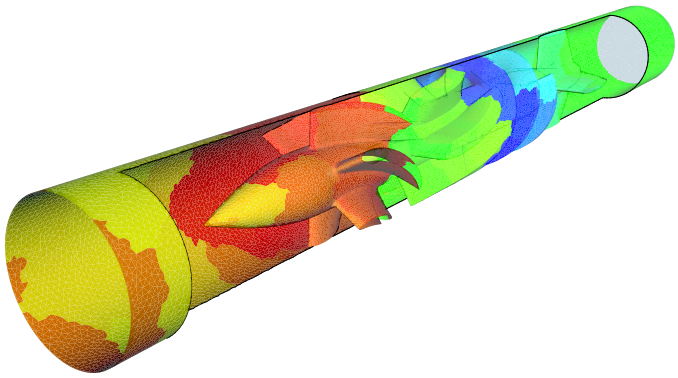


Figure 5. Axial blood pump simulation: partitioning for 1024 processors

with a step size of 0.00005–0.00008 s, representing at least ten full rotations of the impeller. At each time step, four Newton-Raphson iterations are used to solve the nonlinear discretized system, and a GMRES[11] solver with a maximum number of 200 iterations is used to solve the resulting linear system of equations. The computations are performed on an IBM Blue Gene, with up to 4096 CPUs; a sample partitioning of the domain for 1024 processors is shown in Fig. 5. A sample pressure distribution on the pump surfaces is shown in Fig. 6.

The scalability of the XNS, although satisfactory on partitions smaller than 256 PEs, was found seriously lacking on the IBM Blue Gene platform with over 1024 PEs. Subsequent performance analysis showed that the `MPI_Sendrecv` calls used throughout the XNS all-to-all gather and scatter operations are becoming dominated by zero-sized messages as the number of PEs increases. Such a bottleneck could only be detected when employing very large number of PEs, complicating the task of pinpointing the problem which was also specific to the Blue Gene architecture. After introducing conditional and separate `MPI_Send` and `MPI_Recv` calls the scalability improved dramatically, with the

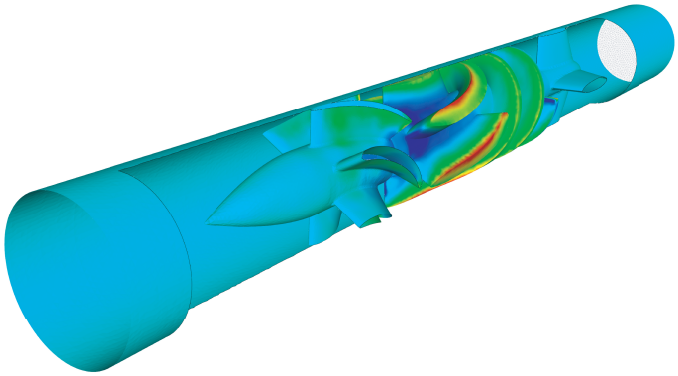


Figure 6. Axial blood pump simulation: sample pressure distribution

performance increasing almost sixfold on 4096 PEs, as shown in Fig. 7.

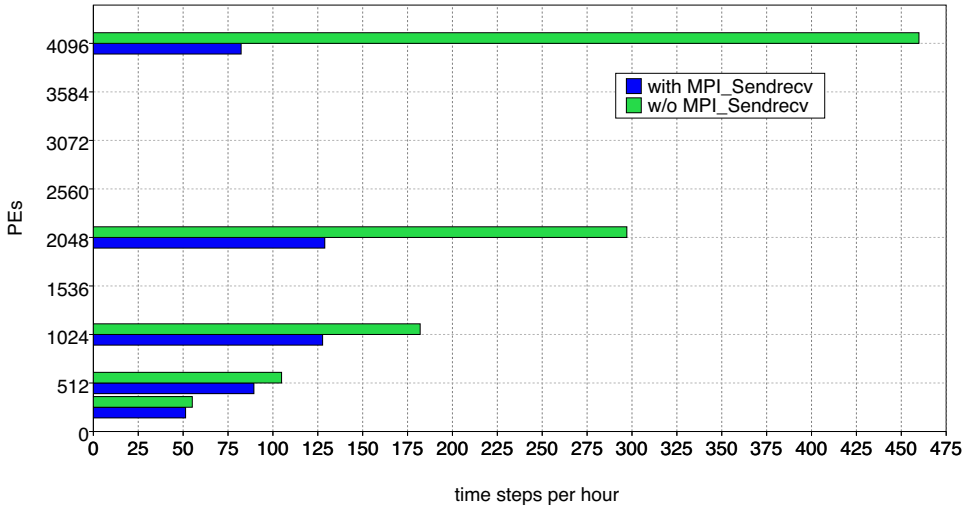


Figure 7. Scalability of the XNS code with and without `MPI_Sendrecv`

In view of the need for design optimization, where unsteady flow fields as well as their sensitivities with respect to the design parameters must be computed repeatedly while seeking a minimum of an flow-dependent objective function, the use of thousands of CPUs is a critical factor that makes such optimization practical. Authors' experience shows that the use of 4096 PEs results in computational speeds in excess of 450 time steps per hour for the model problem, which means that a complete 10-revolution computation can be accomplished in less than 3 hours of wall-clock time.

Acknowledgments

The authors are indebted to the Helmholtz Young Investigator Group "Performance Analysis of Parallel Programs" at the Research Centre Jülich, in particular to Prof. Felix Wolf and Dr. Brian Wylie, for the help in identifying the performance bottlenecks limiting the scalability on the Blue Gene and for the use of the Scalasca (`http://www.scalasca.org`) tool. The benchmark computations were performed with a grant of computer time provided by the Research Centre Jülich.

References

1. M. Yoshikawa, K. Nonaka, J. Linneweber, S. Kawahito, G. Ohtsuka, K. Nakata, T. Takano, S. Schulte-Eistrup, J. Glueck, H. Schima, E. Wolner, and Y. Nosé, *Development of the NEDO implantable ventricular assist device with gyro centrifugal pump*, Artificial Organs, **24**, 459–467, (2000).
2. M. H. Kroll, J. D. Hellums, L. V. McIntire, A. I. Schafer, and J. L. Moake, *Platelets and shear stress*, Blood, **85**, 1525–1541, (1996).
3. J. D. Hellums, *1993 Whitaker Lecture: Biorheology in thrombosis research*, Annals of Biomedical Engineering, **22**, 445–455, (1994).
4. M. Behr and T. E. Tezduyar, *Finite element solution strategies for large-scale flow simulations*, Computer Methods in Applied Mechanics and Engineering, **112**, 3–24, (1994).
5. T.E. Tezduyar, M. Behr, and J. Liou, *A new strategy for finite element computations involving moving boundaries and interfaces – the deforming-spatial-domain/space-time procedure: I. The concept and the preliminary tests*, Computer Methods in Applied Mechanics and Engineering, **94**, 339–351, (1992).
6. T. J. R. Hughes, W. K. Liu, and T. K. Zimmermann, *Lagrangian-Eulerian finite element formulation for incompressible viscous flows*, Computer Methods in Applied Mechanics and Engineering, **29**, 329–349, (1981).
7. M. Behr and T. E. Tezduyar, *Shear-Slip Mesh Update Method*, Computer Methods in Applied Mechanics and Engineering, **174**, 261–274, (1999).
8. J. G. Kennedy, M. Behr, V. Kalro, and T. E. Tezduyar, *Implementation of implicit finite element methods for incompressible flows on the CM-5*, Computer Methods in Applied Mechanics and Engineering, **119**, 95–111, (1994).
9. M. Behr, D. Arora, N. A. Benedict, and J. J. O'Neill, *Intel compilers on Linux clusters*, Technical report, Intel Corporation, Hillsboro, Oregon, (2002).
10. S. Carney, M. A. Heroux, G. Li, R. Pozo, K. A. Remington, and K. Wu, *A revised proposal for a sparse BLAS toolkit*, Preprint 94-034, Army High Performance Computing Research Center, Minneapolis, Minnesota, (1994).
11. Y. Saad and M. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal of Scientific and Statistical Computing, **7**, 856–869, (1986).